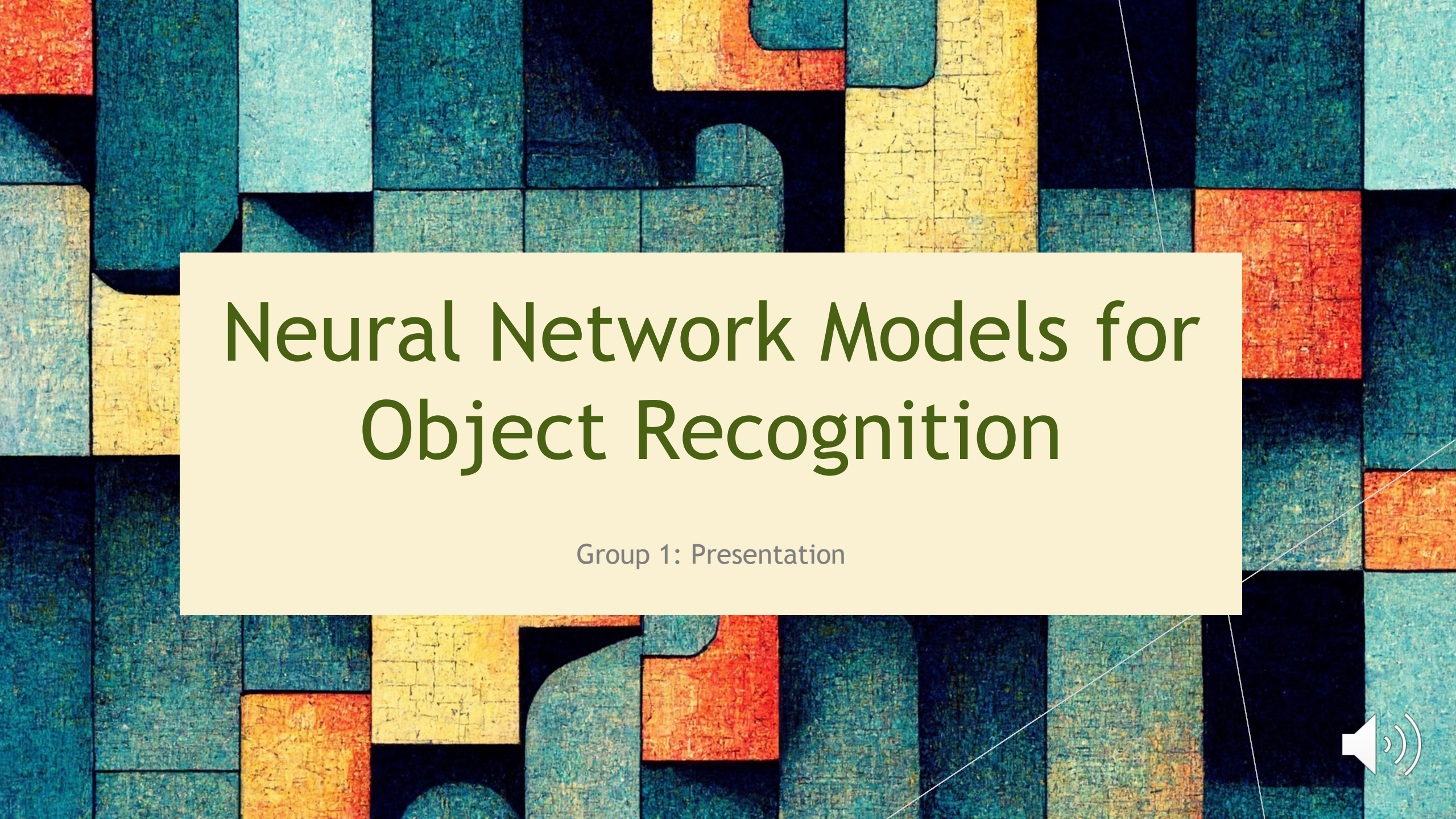


Neural Network Models for Object Recognition

Group 1: Presentation

Contents

Introduction

Object recognition - deep learning application

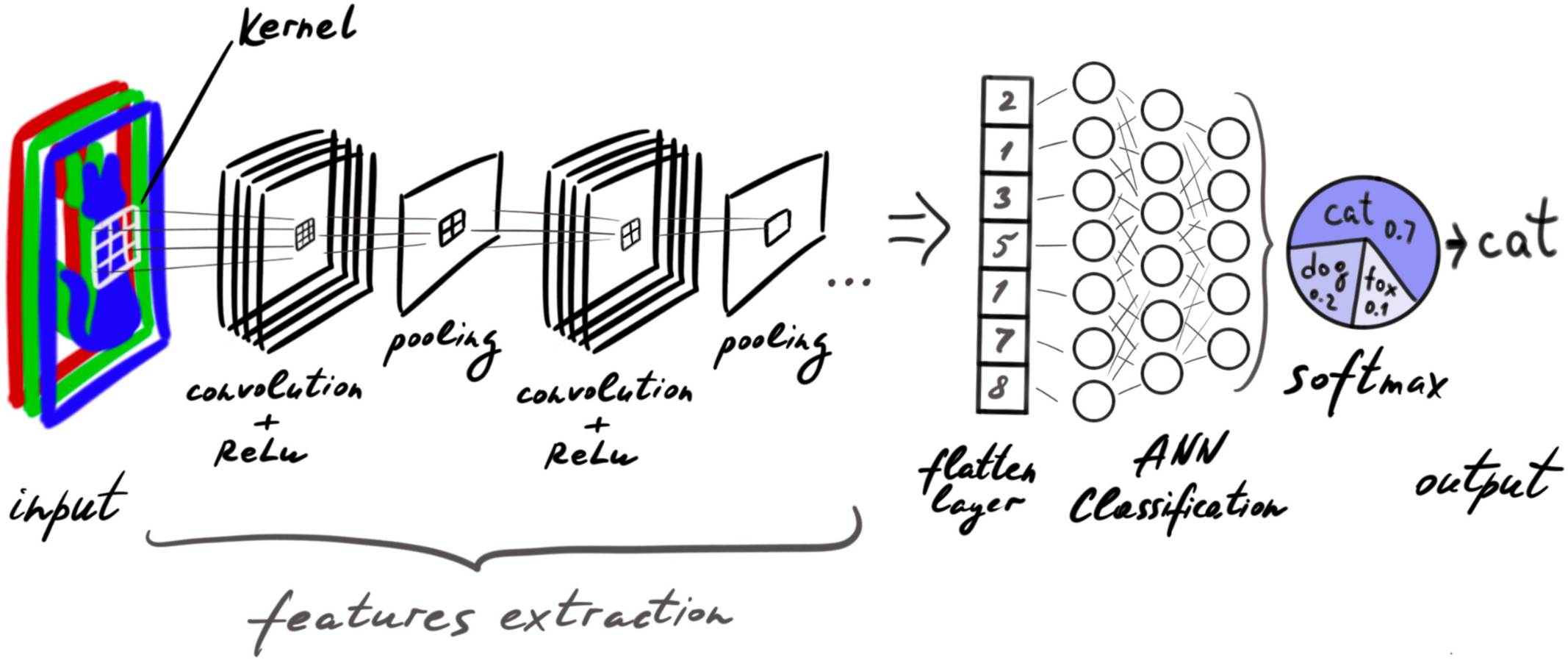
Machines perceive like humans - drive cars, read and comprehend documents

Convolution neural network (CNN) - object detection through feature learning

Artificial neural network (ANN) - Classify object

Neural Network Design

Two main steps: Feature learning and classification



Aim

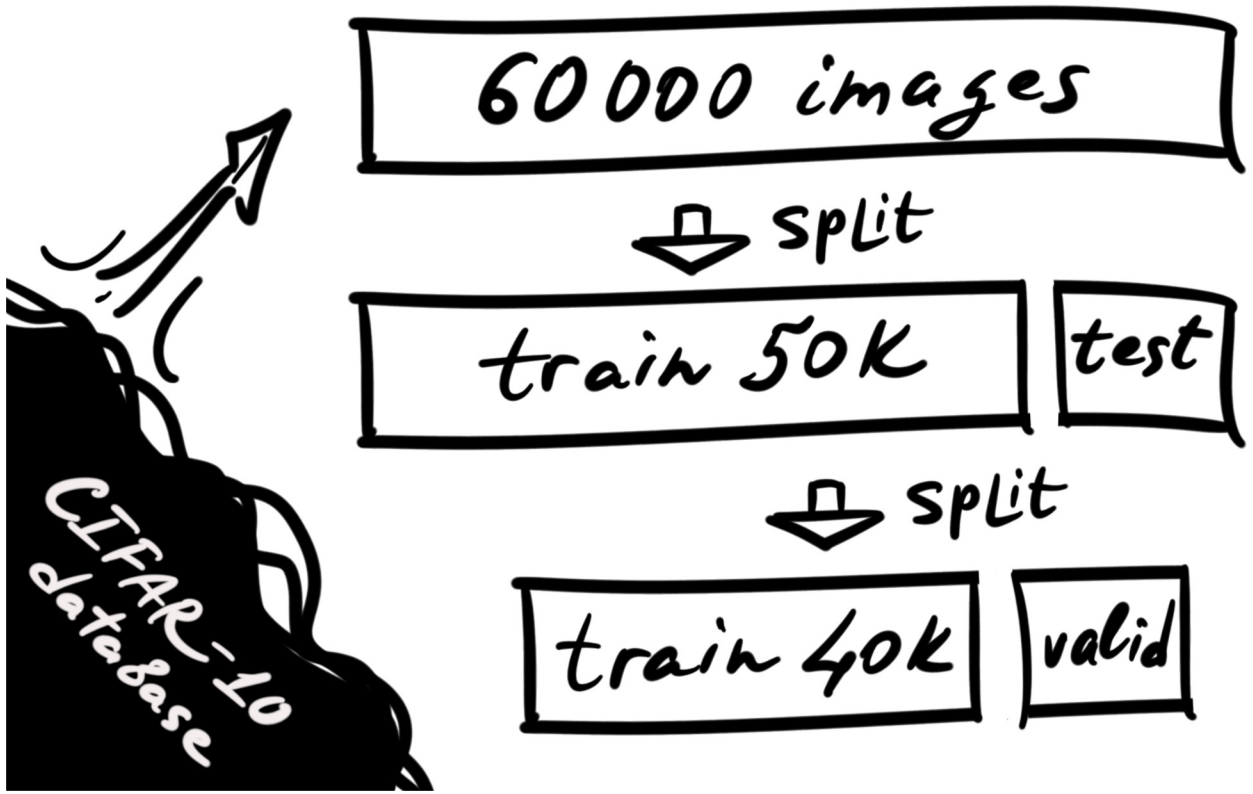
To build a neural network for object recognition

To evaluate the built model

Image data: Open source CIFAR-10- Object Recognition image dataset, hosted in Kaggle. It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. 10 classes: airplane, automobile, bird, cat, dear, dog, frog, horse, ship, truck

Python libraries used: numpy, tensorflow, keras, pandas, seaborn,

Data split



The CIFAR-10 dataset has 50000 training images and 10000 test images. 10000 images out of 50000 training images were used for validation.

Training:Validation:Testing = 40000:10000:10000

Training data: Model trained the data
Validation data: Model fine-tuning, help to improve model performance
Testing data: Provides measure of accuracy of built model

Preprocessing - Normalising the data

```
In [26]: x_train = x_train / 255.0

In [27]: x_test =  x_test / 255.0
```

```
In [21]: x_train[0]

Out[21]: array([[[ 59,  62,  63],
                 [ 43,  46,  45],
                 [ 50,  48,  43],
                 ...,
                 [158, 132, 108],
                 [152, 125, 102],
                 [148, 124, 103]],

                [[ 16,  20,  20],
                 [  0,   0,   0],
                 [ 18,   8,   0],
                 ...,
                 [123,  88,  55],
                 [119,  83,  50],
                 [122,  87,  57]],

                [[ 25,  24,  21],
                 [ 16,   7,   0],
                 [ 49,  27,   8],
                 ...,
                 [118,  84,  50],
                 [120,  84,  50],
                 [109,  73,  42]],

                ...,
```

```
In [58]: x_train

Out[58]: array([[[[0.1372549 , 0.09803922, 0.10196078],
                  [0.10588235, 0.08235294, 0.08235294],
                  [0.09803922, 0.07843137, 0.0745098 ],
                  ...,
                  [0.51764706, 0.50588235, 0.50588235],
                  [0.52156863, 0.4745098 , 0.45490196],
                  [0.49411765, 0.45098039, 0.44313725]],

                 [[0.24705882, 0.21568627, 0.19607843],
                  [0.1254902 , 0.10588235, 0.08235294],
                  [0.06666667, 0.05098039, 0.03137255],
                  ...,
                  [0.4       , 0.37254902, 0.34509804],
                  [0.41176471, 0.34901961, 0.29803922],
                  [0.39215686, 0.3372549 , 0.30196078]],

                 [[0.38823529, 0.35686275, 0.32941176],
                  [0.19215686, 0.17647059, 0.14509804],
                  [0.05882353, 0.04705882, 0.01960784],
```

With Numpy library providing normalisation for every pixel in both training and testing datasets.

From 0 to 255 for each of three channels (RGB) it comes to a range between 0 and 1.

Preprocessing - Encoding to_category for the 10 classes

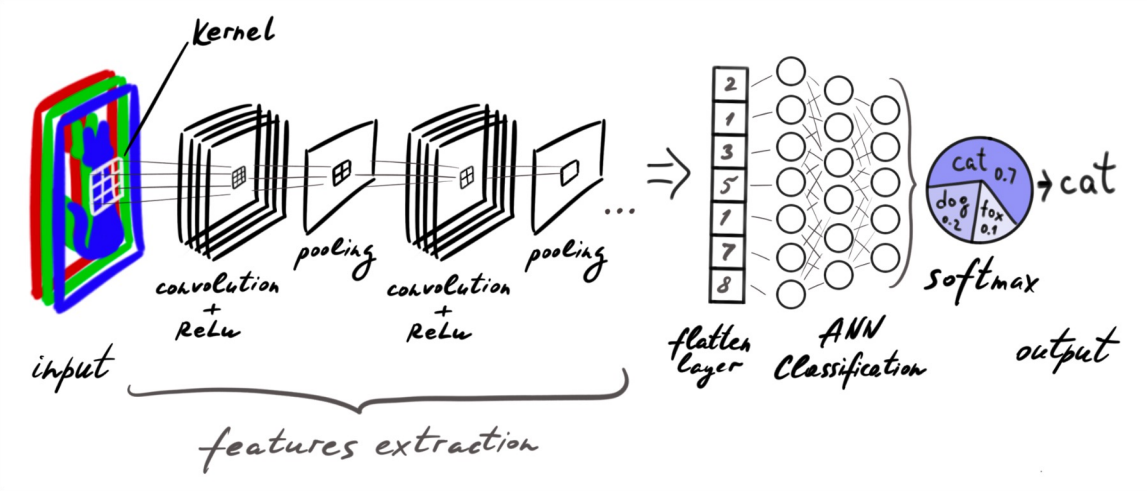
```
y_cat_train_all = to_categorical(y_train_all,10)
```

```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]], dtype=uint8)
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

Model design

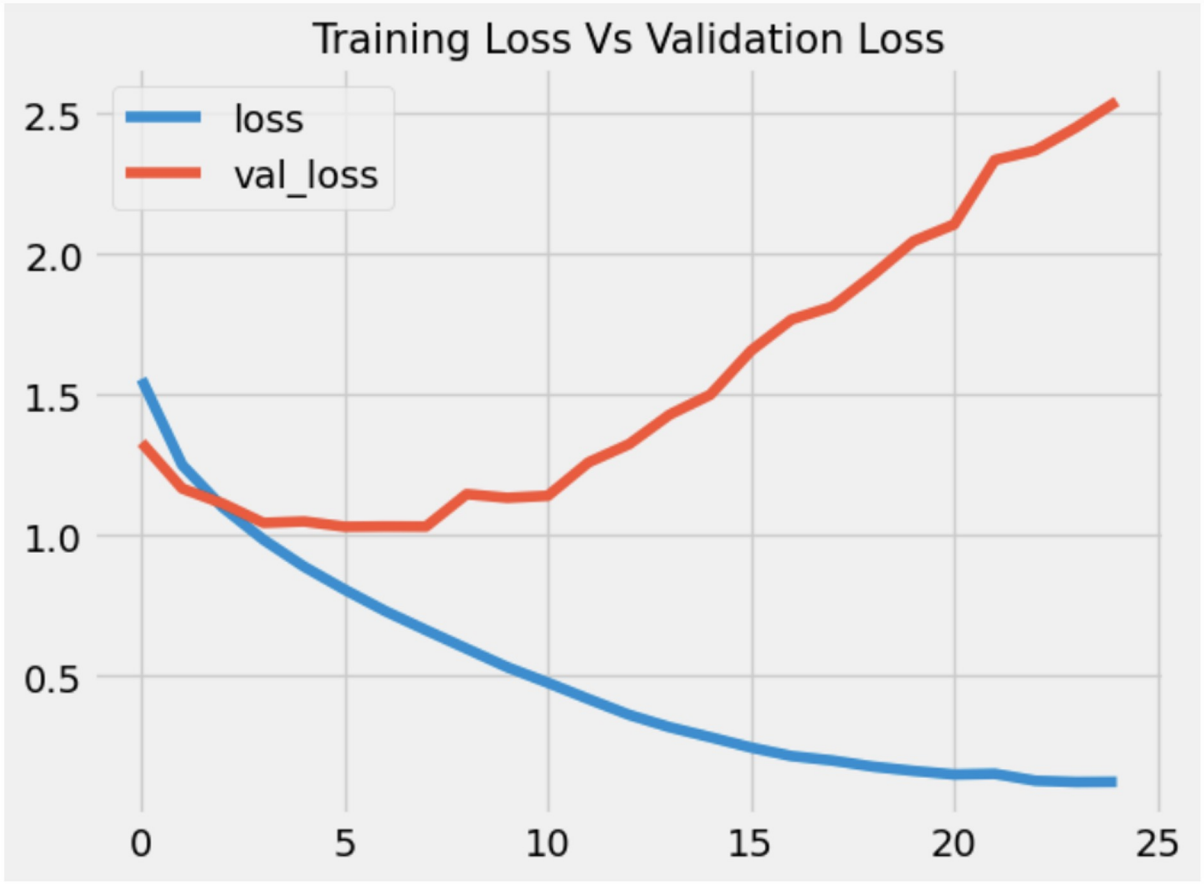
Schematic image for the neural network:



The same written in code:

```
model = Sequential()

# First CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

# Second CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(4,4),activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES
model.add(Flatten())

# Condense
model.add(Dense(256, activation='relu'))

# Classifier
model.add(Dense(10, activation='softmax'))


model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Overfitting alarm



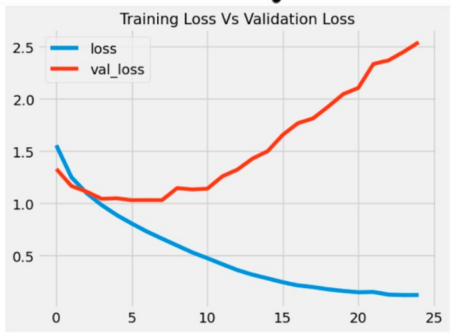
It starts well, but after a few first iterations the loss in validation set increases.

The model shows an overfitting relationship with a very low training loss but relatively high validation loss.

Overfitted neural networks are not good at generalizing. **(Doon et al., 2018)**

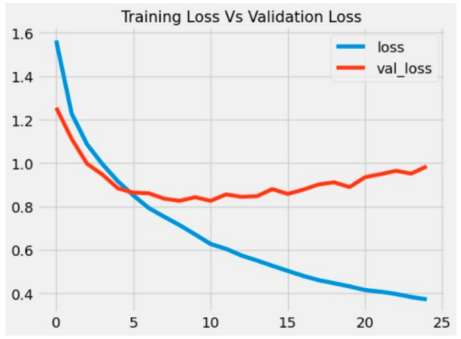
Dropout technique

Without Dropout



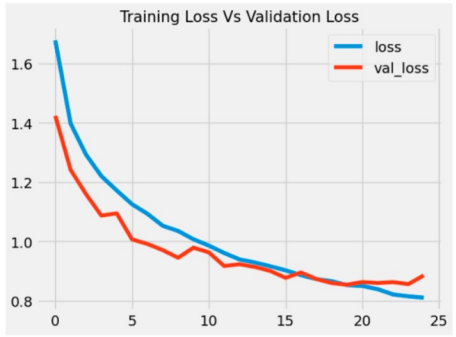
98%

Dropout 0.25



86%

Dropout 0.5



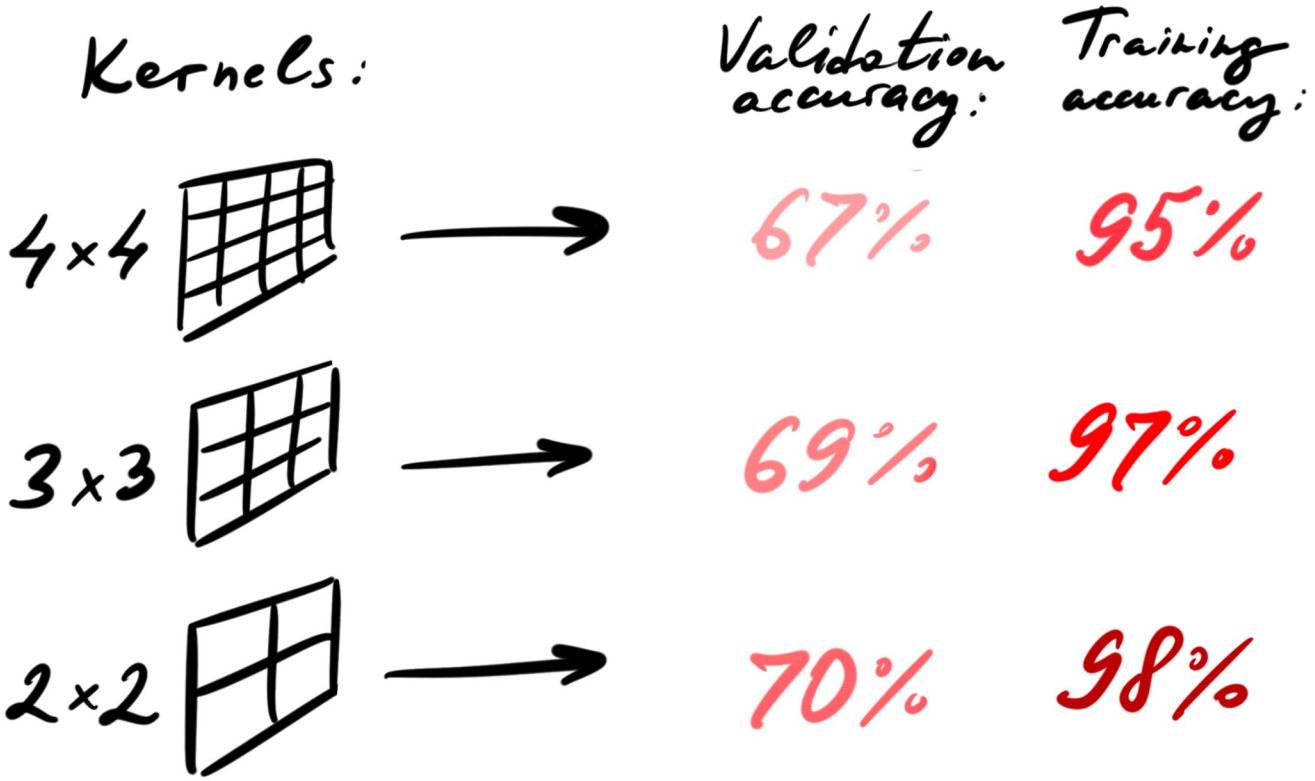
74%

training accuracy

Using the Dropout technique to manage with overfitting.
It randomly drops units from the network with some probability during training. **(Srivastava et al., 2014)**

However on the long distance of many epochs Dropout decreases the training accuracy.

Model evaluation - kernel size



A little correlation of a kernel size and validation/training accuracy of the model.

A relatively small size of the kernel helps to find more details in the picture. The bigger kernel seize could generalise better.

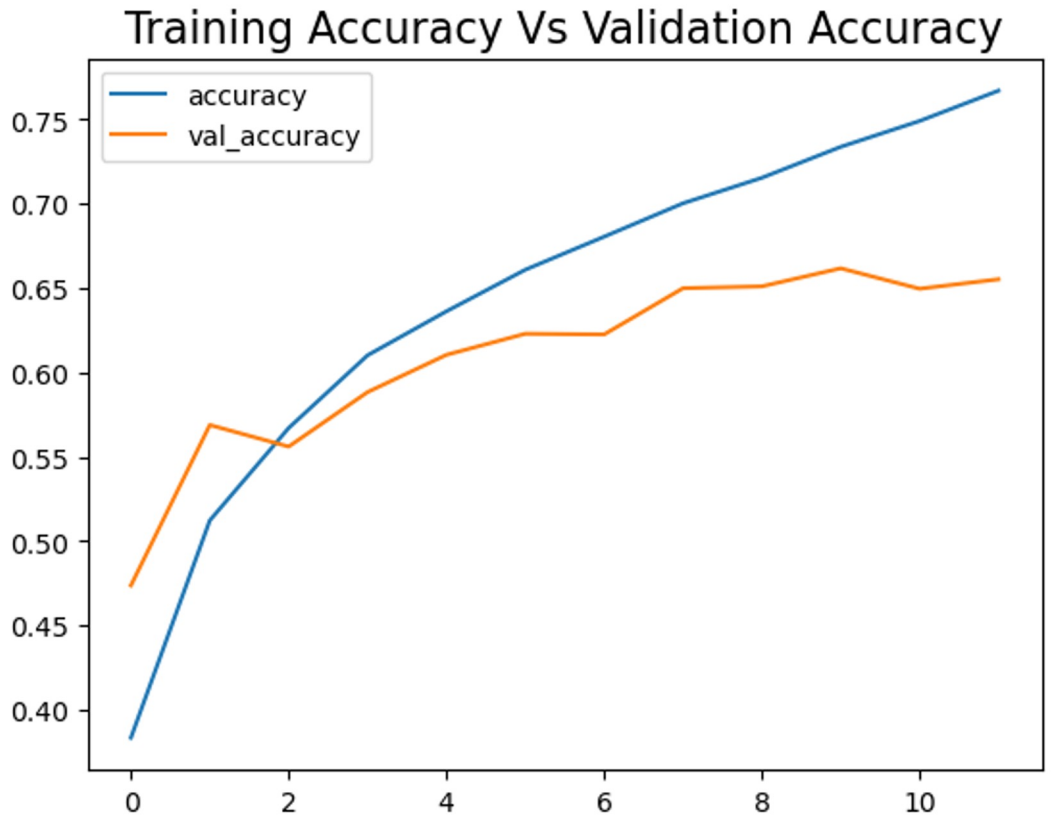
(Cortes et al., 2010)

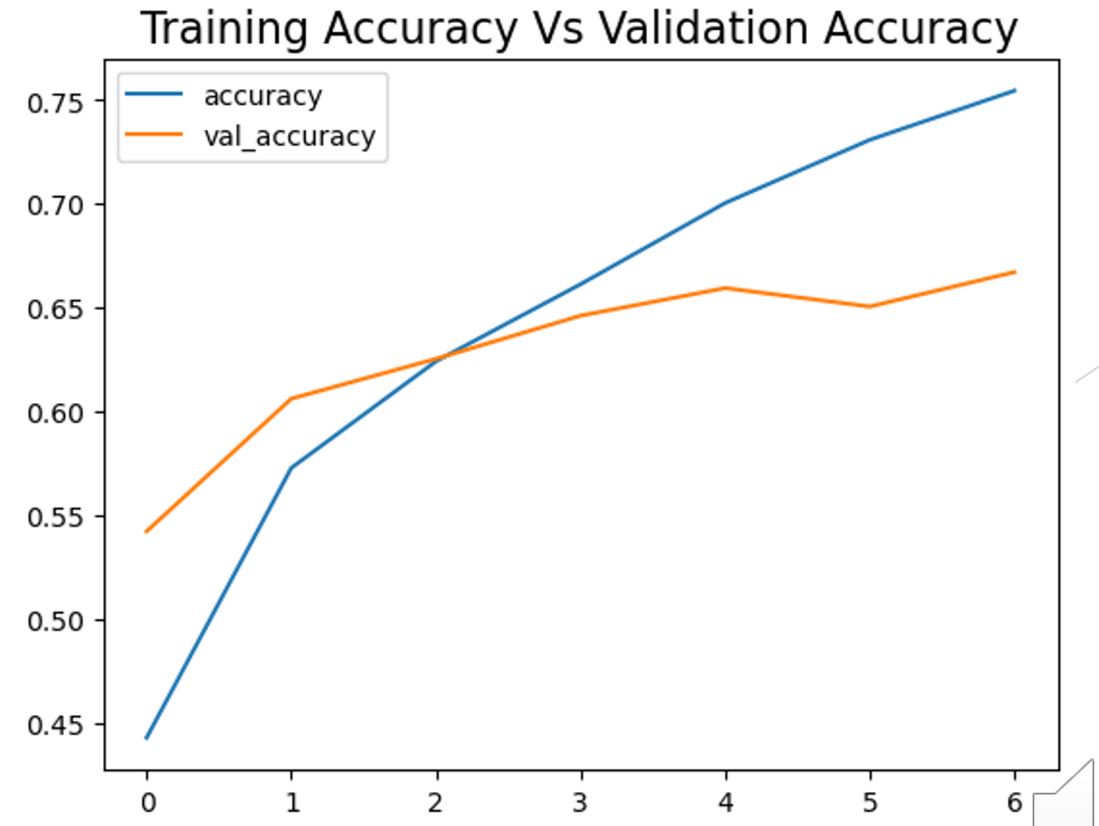
Model evaluation - No of kernels & Early Stopping

Kernels 32 - Training accuracy 75.5%

Kernels 64 - Training accuracy 76.7%

Another way to avoid overfitting





Epochs used

Our model used 12 epochs - optimal

Used early stopping, patience 2

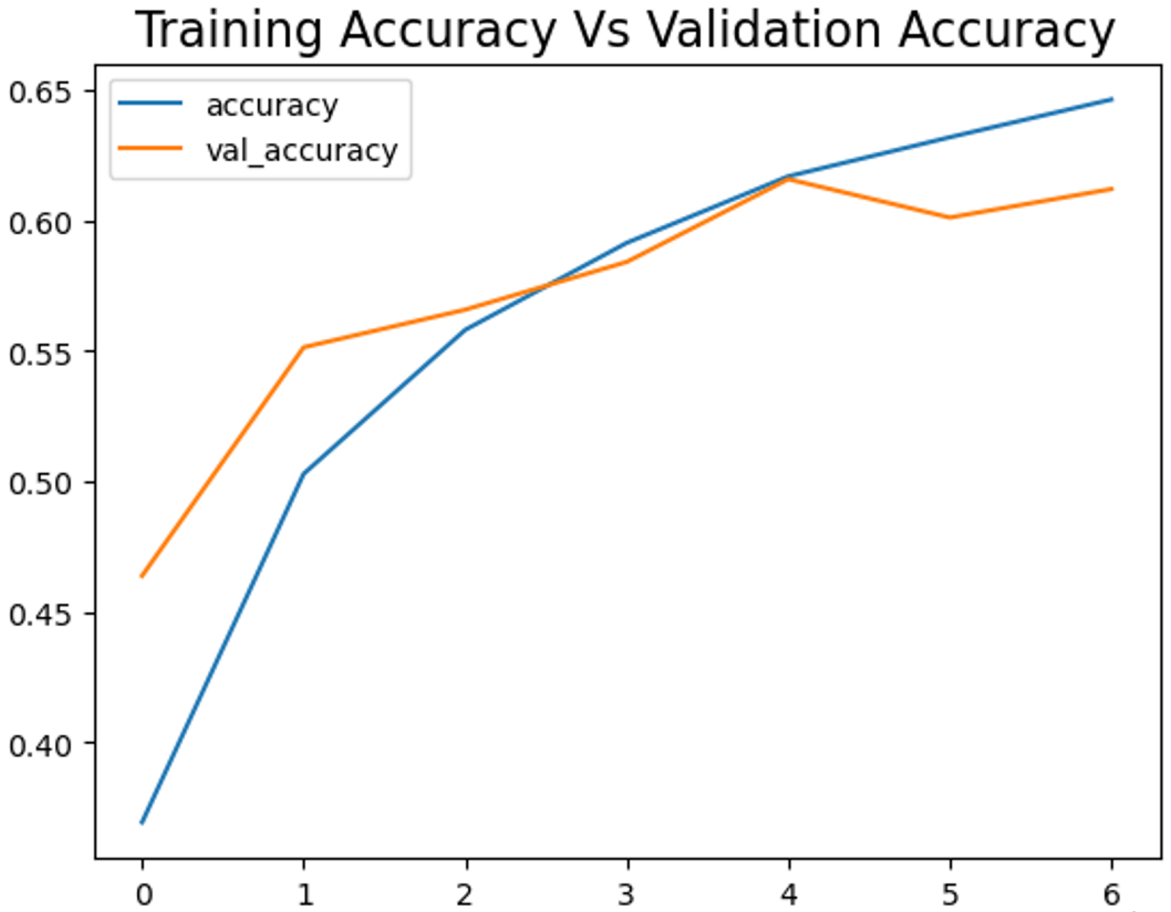
Too many epochs - overfitting problem

	loss	accuracy	val_loss	val_accuracy
0	1.680713	0.383500	1.433848	0.4738
1	1.364287	0.512325	1.236065	0.5689
2	1.221159	0.567125	1.268536	0.5561
3	1.111533	0.610225	1.172620	0.5884
4	1.031733	0.636300	1.119634	0.6105
5	0.966625	0.660950	1.091673	0.6230
6	0.904833	0.680550	1.109121	0.6226
7	0.850928	0.700350	1.017413	0.6500
8	0.802035	0.715475	1.037993	0.6511
9	0.755995	0.733825	1.014872	0.6618
10	0.711462	0.749150	1.078260	0.6497
11	0.662567	0.767125	1.082580	0.6553

Model evaluation- no.of convolution layers

Adding one more convolution layer 2+1

Validation accuracy more closer to training accuracy



Discussion

Key findings:

- The smaller kernel size increases the accuracy for the training model
- Double number of kernels provides better accuracy (+1.2%)
- Dropout helps to manage overfitting, but decreases further accuracy
- One more *Convolutional* + *ReLU* layer increases accuracy
- Early stopping trick prevents from overfitting on the long distance

Conclusion

```
#predictions = model.predict_classes(x_test)
predictions = np.argmax(model.predict(x_test), axis=-1)
```

```
313/313 [==============================] - 1s 3ms/step
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.72      0.77      0.74      1000
           1       0.81      0.81      0.81      1000
           2       0.65      0.60      0.62      1000
           3       0.50      0.61      0.55      1000
           4       0.67      0.66      0.67      1000
           5       0.65      0.52      0.58      1000
           6       0.81      0.75      0.78      1000
           7       0.75      0.79      0.77      1000
           8       0.81      0.82      0.81      1000
           9       0.79      0.77      0.78      1000

    accuracy                           0.71     10000
   macro avg       0.71      0.71      0.71     10000
weighted avg       0.71      0.71      0.71     10000
```

The CIFAR-10 data set is divided into a training and a test set, but does not come with a validation set by default.

We created a validation set, then we trained the model on the training set, and tested on the validation set to see if it is a good fit.

In the end the testing set was used to calculate accuracy: **about 71%.**

References

- The CIFAR-10 dataset. Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 1 Dec 2022]
- Krizhevsky, A. & Hinton, G. (2009) Learning multiple layers of features from tiny images.
- Recht, B., Roelofs, R., Schmidt, L. and Shankar, V. (2018) Do cifar-10 classifiers generalize to cifar-10?. arXiv preprint arXiv:1806.00451.
- Doon, R., Rawat, T.K. and Gautam, S. (2018). Cifar-10 classification using deep convolutional neural network. In 2018 IEEE Punecon (pp. 1-5). IEEE.
- Krizhevsky, A. and Hinton, G. (2010) Convolutional deep belief networks on cifar-10. Unpublished manuscript, 40(7), pp.1-9.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), pp.1929-1958.
- Cortes, C., Mohri, M. and Talwalkar, A. (2010) On the impact of kernel approximation on learning accuracy. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 113-120). JMLR Workshop and Conference Proceedings.

Code

- Please find the code added at the end of the transcript